Processadores AVR

Ronaldo Husemann

Família AVR

AVR

Advanced Virtual RISC,
cujos fundadores são Alf Egil Bogen, Vegard Wollan RISC

A arquitetura AVR foi concebida por dois estudantes do Norwegian Institute of Technology (NTH) e posteriormente refinada e desenvolvida pela **Atmel Norway**, companhia fundada por dois arquitetos de chip

Família AVR

Registradores:

32 registradores de 8 bits de propósito geral
Chamados R0,R1,R2 a R31.

Tres registradores compostos

Program counter
Stack Pointer
Status Register

Família AVR

Menor chip -> menor consumo de energia

- Arquitetura RISC de 8 bits
- Pipeline simples (busca & execução)
  Instrucoes em um ciclo
  8MHz = 8MIPS .
- 32 registradores genéricos
- ULA sem dependencia com acumulador
- 3 pares de registradores de indice
- Registradores & IO são mapeados em SRAM

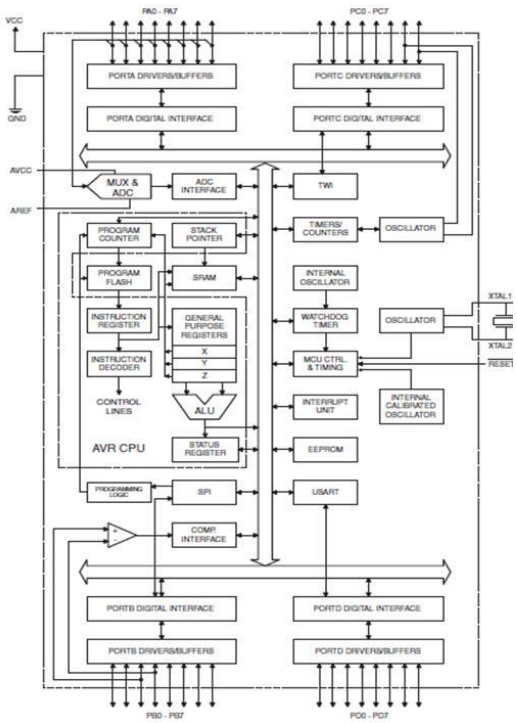
Tempo total de busca e execução é 2 CLKs (um pra buscar e outro pra executar).
No PIC -> 8 CLKs

O número de instruções executadas corresponde ao número de Hz da CPU

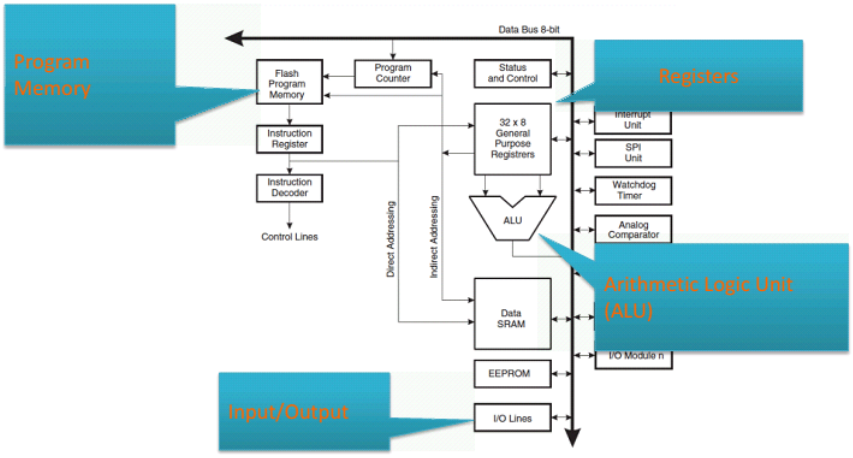
Um salto esvazia o pipeline e gasta 2 CLKs, no PIC gasta 8 CLKs

Recursos de Hardware

- Oscilador/clock interno ou externo
- Detector de Brown Out
- Timers
- Dois um mais PWM
- Uma ou mais USART
- I2C
- Real time clock
- ADC de 10 bits
- Comparador analógico
- Interrupções externas
- Captura de tempo de pulso
- EEPROM
- USB/CAN/RF



AVR



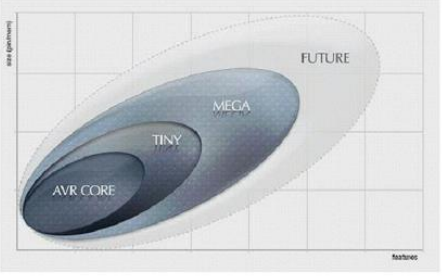
Família AVR

Microcontroladores da Família AVR tem uma arquitetura simples

→mesmo código é aceito para todas famílias

→ código de 1 Kbytes a 256 Kbytes

→8 a 100 pinos



Família AVR

AVR Clássico (AT90Sxxxx)

Modelo original AVR posteriormente aprimorado.

Mega AVR (ATmegaxxxx)

Versão mais poderosa incluindo mais de 120 instruções e vários periféricos extra.

Memória de programa: 4K a 256K bytes

Encapsulamento: 28 a 100 pinos

Tiny AVR (ATtinyxxxx)

Voltada para aplicações de baixo custo e consumo de potencia .

Memória de programa: IK a 8K bytes

Encapsulamento: 8 a 28 pinos

Special purpose AVR

Desenvolvidos para aplicações e capacidades especiais: USB controller, CAN controller, LCD controller, Zigbee, Ethernet controller, FPGA, e ou advanced PWM.

Família AVR

Table 1-2: Some Members of the Classic Family

Part Num.	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
AT90S2313	2K	128	128	15	0	2	SOIC20, PDIP20
AT90S2323	2K	128	128	3	0	1	SOIC8, PDIP8
AT90S4433	4K	128	256	20	6	2	TQFP32, PDIP28

Table 1-3: Some Members of the ATmega Family

Part Num.	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
ATmega8	8K	1K	0.5K	23	8	3	TQFP32, PDIP28
ATmega16	16K	1K	0.5K	32	8	3	TQFP44, PDIP40
ATmega32	32K	2K	1K	32	8	3	TQFP44, PDIP40
ATmega64	64K	4K	2K	54	8	4	TQFP64, MLF64
ATmega1280	128K	8K	4K	86	16	6	TQFP100, CBGA

Notes:

1. All ROM, RAM, and EEPROM memories are in bytes.
2. Data RAM (general-purpose RAM) is the amount of RAM available for data manipulation (scratch pad) in addition to the register space.
3. All the above chips have USART for serial data transfer.

Família AVR

Table 1-4: Some Members of the Tiny Family

Part Num.	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
ATtiny13	1K	64	64	6	4	1	SOIC8, PDIP8
ATtiny25	2K	128	128	6	4	2	SOIC8, PDIP8
ATtiny44	4K	256	256	12	8	2	SOIC14, PDIP14
ATtiny84	8K	512	512	12	8	2	SOIC14, PDIP14

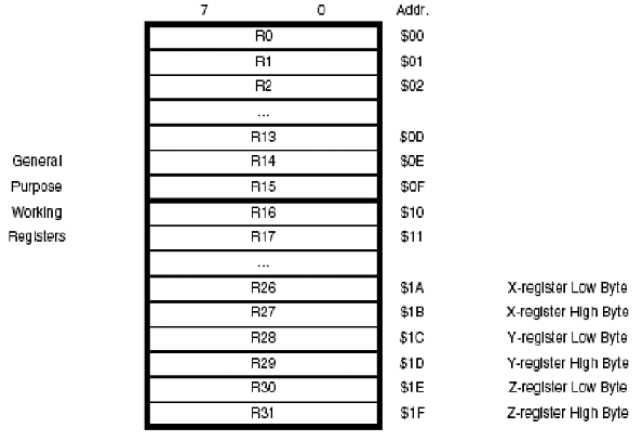
Embedded Systems

Table 1-5: Some Members of the Special Purpose Family

Part Num.	Code ROM	Data RAM	Data EEPROM	Max I/O pins	Special Capabilities	Timers	Pin numbers & Package
AT90CAN128	128K	4K	4K	53	CAN	4	LQFP64
AT90USB1287	128K	8K	4K	48	USB Host	4	TQFP64
AT90PWM216	16K	1K	0.5K	19	Advanced PWM	2	SOIC24
ATmega169	16K	1K	0.5K	54	LCD	3	TQFP64, MLF64

Família AVR

Arquitetura de registradores



Família AVR

Possuem arquitetura Harvard:

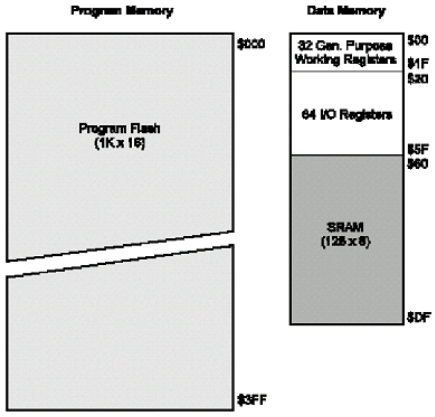
Arquitetura que tem barramentos de código e dados fisicamente separados

memória de programa (Flash)

Memória de dados (sram)

Também usada pelo PIC

Vantagem: um acesso à memória consegue ler a instrução completa (devido à largura de 16 bits). Diferente do MCS51



Família AVR

Arquitetura de instruções

Sem dependencia de acumulador

Exemplo

add R23, R11

codificado como opcode de 16 bits 0x0EEB.

Padrão em binário: 0000 1110 1110 1011

6 bits 00011 definem a instrução *add* .
5 bits 10111 indicam primeiro operador = *register 23*
5 bits 01011 indicam segundo operador = *register 11*

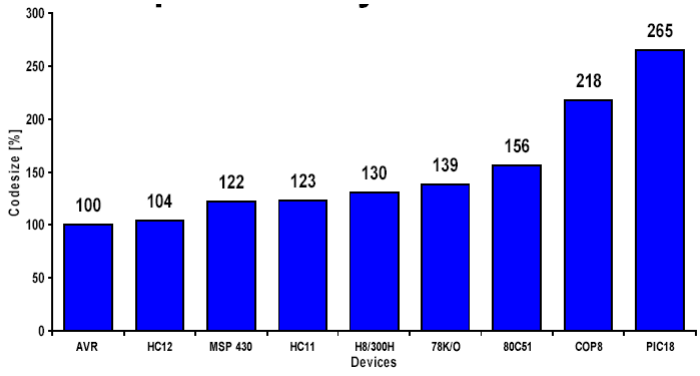
Exemplo de código

```
int main(void)
{
    char i;
    char j;
    i=0;
    j=0;
    while (i<10) {
        j = j + i;
        PORTB = j;
        PORTB = i;
        i++;
    }
    return 0;
}
```

```
000000be <main>:
  be:   80 e0   ldi   r24, 0x00   ; 0
  c0:   90 e0   ldi   r25, 0x00   ; 0
  c2:   98 0f   add   r25, r24
  c4:   92 bb   out   0x18, r25   ; 18
  c6:   82 bb   out   0x18, r24   ; 18
  c8:   8f 5f   subi  r24, 0xFF   ; 255
  ca:   8a 30   cpi   r24, 0x0A   ; 10
  cc:   d1 f7   brne  .-12        ; 0xc2 <main+0x4>
  ce:   80 e0   ldi   r24, 0x00   ; 0
  d0:   90 e0   ldi   r25, 0x00   ; 0
  d2:   08 95   ret
```

Família AVR

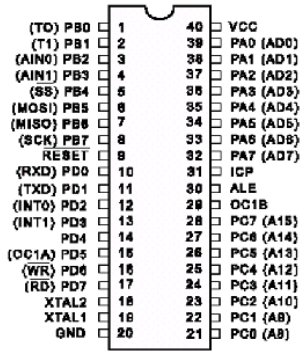
Comparação entre tecnologias em termos de tamanho de código



16

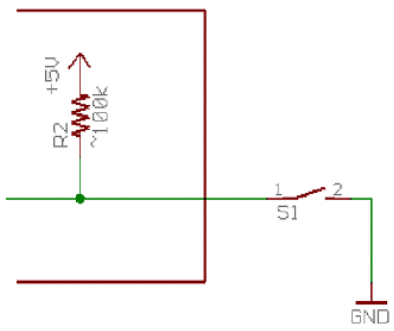
AVR AT90S8515 Pinout

- Portas de propósito geral
  - PORTA
  - PORTB
  - PORTC
  - PORTD
- Pinos especiais
  - Crystal (XTAL1/XTAL2)
  - RESET
  - ICP, OLE, OC1B
- Alimentação (VCC/GND)



Circuito de I/O

- Chave simples sem necessidade de componentes externos



Família AVR

Pinos de I/O

Port B Data Register – PORTB -> Escrita

Bit	7	6	5	4	3	2	1	0	
$18 ($38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

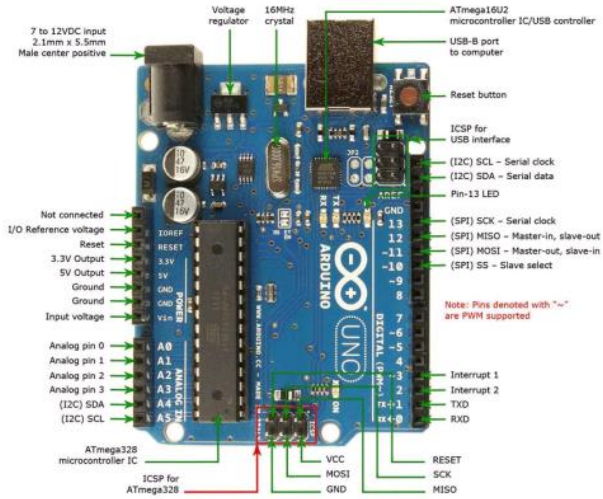
Port B Data Direction Register– DDRB

Bit	7	6	5	4	3	2	1	0	
$17 ($37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

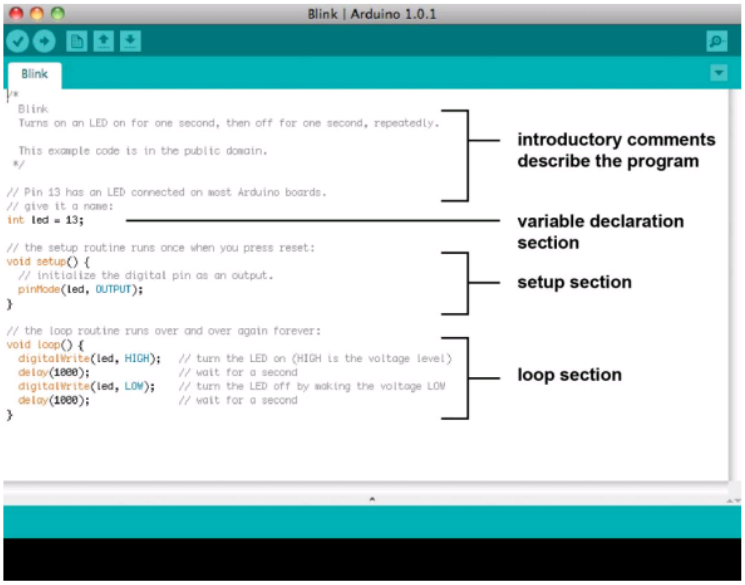
Port B Input Pins Address –PINB -> Leitura

Bit	7	6	5	4	3	2	1	0	
$16 ($36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Uso comum de AVRs



Uso comum de AVRs



Exemplo

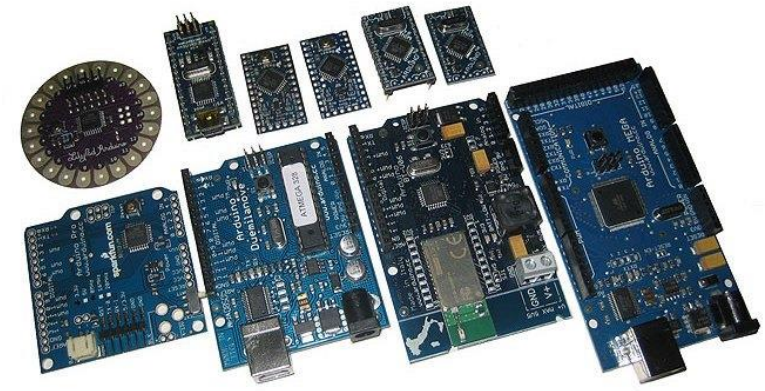
```
void setup()
{
pinMode(13, OUTPUT);   //configura pino 13 como saida
}

// Pisca LED
void loop()
{
digitalWrite(13,HIGH);      // Ativa LED
delay(100);                 // Aguarda 0,1 s
digitalWrite(13,LOW);       // Apaga LED
delay(100);                 // Aguarda 0,1 s
}
```

Exemplo

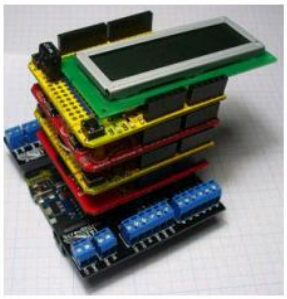
```
const int inputPin = 2;

void setup()
{
pinMode(inputPin, INPUT); //configura pino 2 como entrada
}

// Le pino
void loop()
{
int val = digitalRead(inputPin); // le pino de entrada

}
```

Uso comum de AVRs

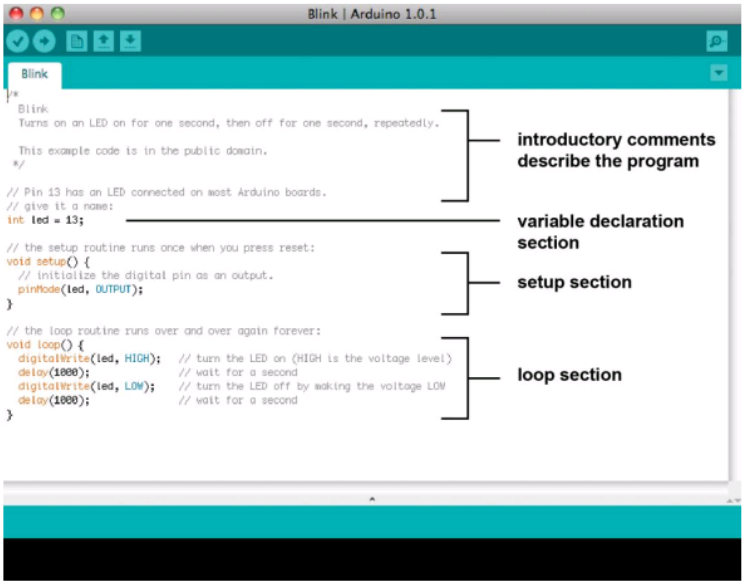
- Diferentes placas



Uso comum de AVRs



Uso comum de AVRs



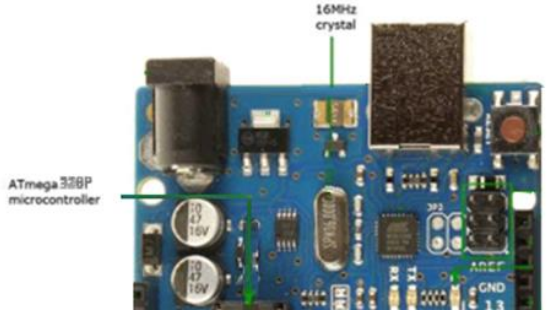
Programa de teste Arduino

```
void setup()
{
pinMode(13, OUTPUT);   //configura pino 13 como saida
}

// Pisca LED
void loop()
{
digitalWrite(13,HIGH);     // Ativa LED
delay(500);                // Aguarda 0,5 s
digitalWrite(13,LOW);      // Apaga LED
delay(500);                // Aguarda 0,5 s
}
```

Exemplo

- Onda de 300Hz

Placa Arduino UNO
Microcontrolador ATMEGA328P
Frequencia da CPU: 16MHz



Exemplo

- Onda de 300Hz

T = 1/300 = 0,0033 s = 3,33 ms

2 CLK pra execução -> T/2 = 3,33/2 = 1,66 ms alto e 1,66 ms baixo

Placa Arduino UNO
Microcontrolador ATMEGA328P
Frequencia da CPU: 16MHz

Freq_LED 300 Hz
T_LED = 1/300 = 3,3ms

Exemplo

- Onda de 300Hz

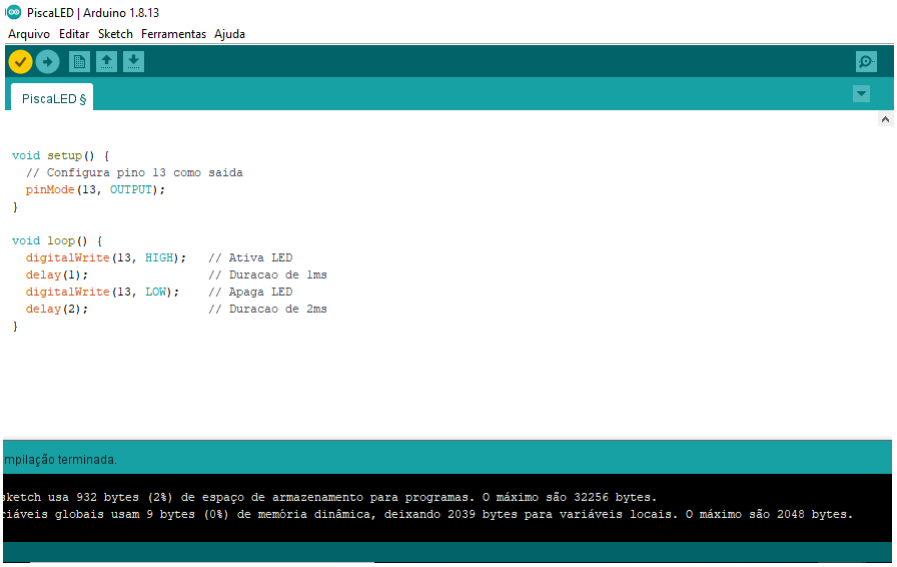
Placa Arduino UNO
Microcontrolador ATMEGA328P
Frequencia da CPU: 16MHz

Freq_LED 300 Hz
T_LED = 1/300 = 3,3ms → **1ms**
→ **2ms**

Exemplo – Onda de 300Hz

```
void setup()
{
pinMode(13, OUTPUT);   //configura pino 13 como saida
}

// Pisca LED
void loop()
{
digitalWrite(13,HIGH);      // Ativa LED
delay(1);                   // Duração de 1ms
digitalWrite(13,LOW);       // Apaga LED
delay(2);                   // Duração de 2ms
}
```

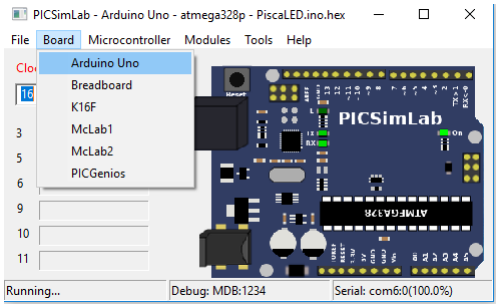
Exemplo – Onda de 300Hz



Exemplo

- Testando no PICSimLab

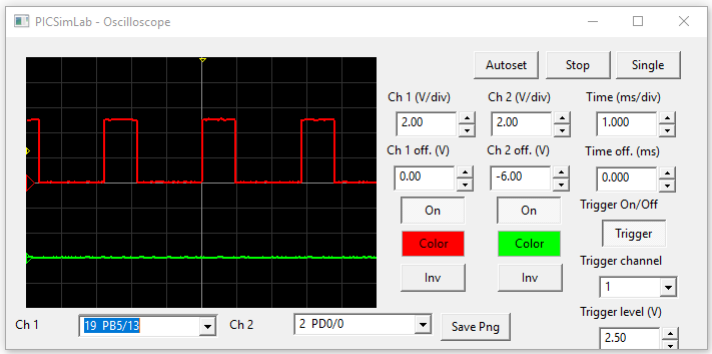
Placa Arduino UNO
Microcontrolador ATMEGA328P
Frequencia da CPU: 16MHz



Simulador

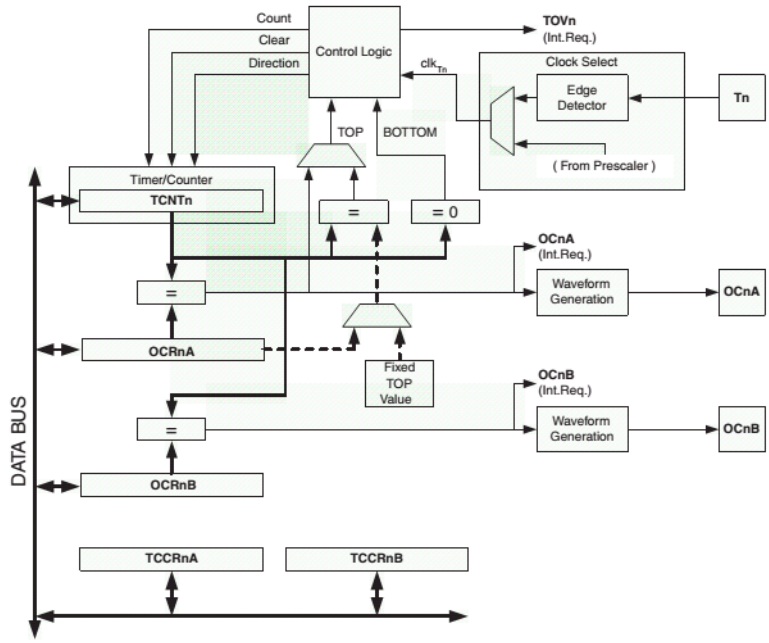
- Testando no PICSimLab

<usr dir>**/local/App/arduino1238/proj.hex**



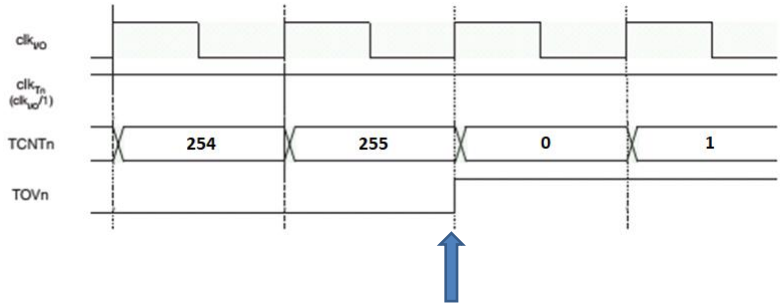
AVR Timer/Counter 0

- Temporizador de 8 bits
  - conta de 0 a 255 (0xFF)
  - Fonte de Clock interna ou externa
- Prescaler
- Interrupção no Overflow
- Comparadores (A e B) para gerar forma de onda (PWM)



AVR Timer/Counter 0

- Modo normal de operação

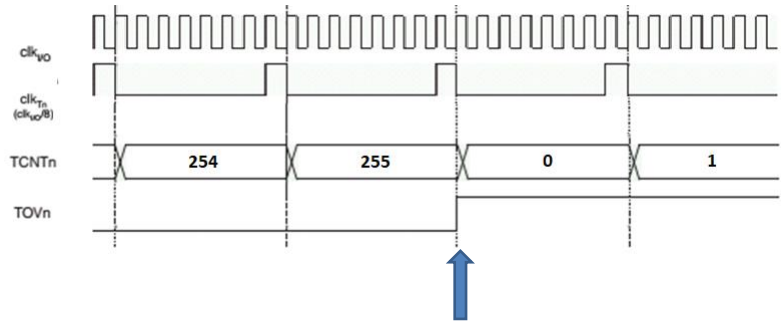


TOV0 é ativado após overflow

AVR Timer/Counter 0

- Modo normal de operação

Exemplo com prescaler = 8

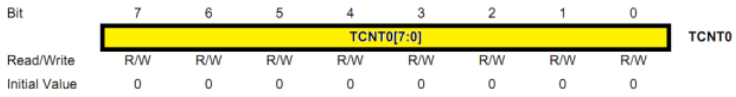


TOV0 é ativado após overflow

AVR Timer/Counter 0

TCNT0 Register

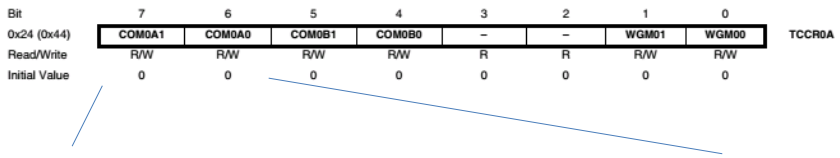
Timer/Counter Register – TCNT0



AVR Timer/Counter 0

TCCR0A

Timer/Counter Control A Register – TCCR0A



COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

AVR Timer/Counter 0

TCCR0A

Timer/Counter Control A Register – TCCR0A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[1][2]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	TOP	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Note: 1. MAX = 0xFF
2. BOTTOM = 0x00

AVR Timer/Counter 0

TCCR0B

Timer/Counter Control B Register – TCCR0B

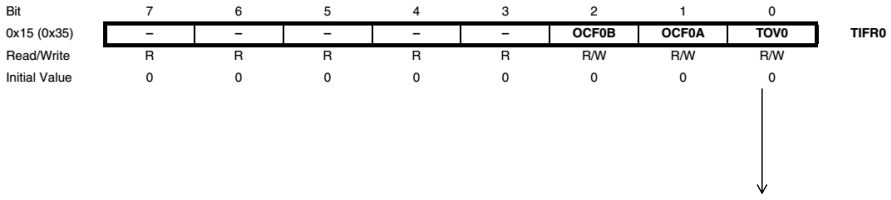
Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

AVR Timer/Counter 0

TIFR0

Timer/Counter 0 Interrupt Flag Register



Pra zerar o pino é necessário colocar 1

TOV0 is cleared by writing a logic one to the flag.

Exemplo

- Onda de 300Hz

Placa Arduino UNO
Microcontrolador ATMEGA328P
Frequencia da CPU: 16MHz = CLK do timer

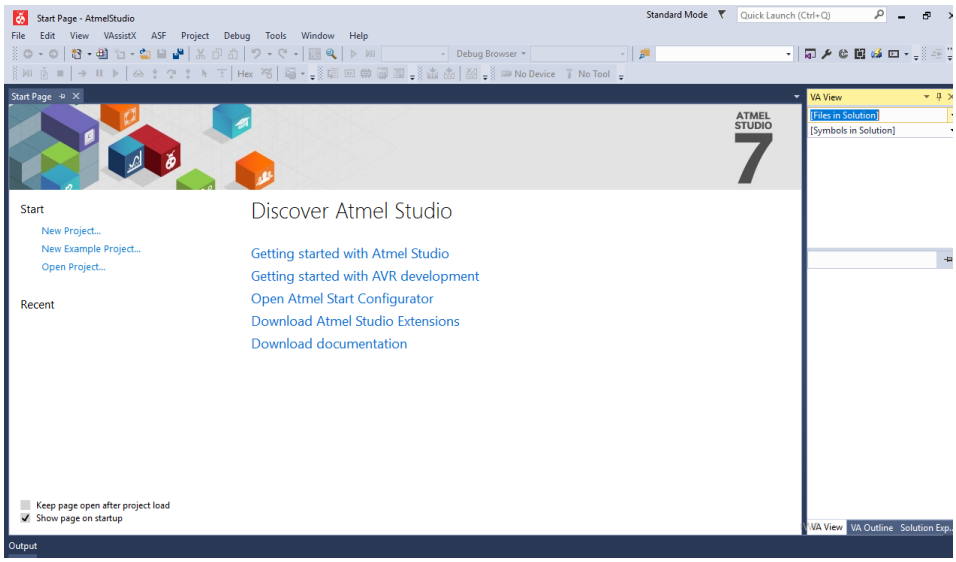
No MCS51: CLK timer = CLK CPU/12
No PIC: CLK timer = CLK CPU/4

Freq_LED 300 Hz
T_LED = 1/300 = 3,3ms => 1,66ms x 2

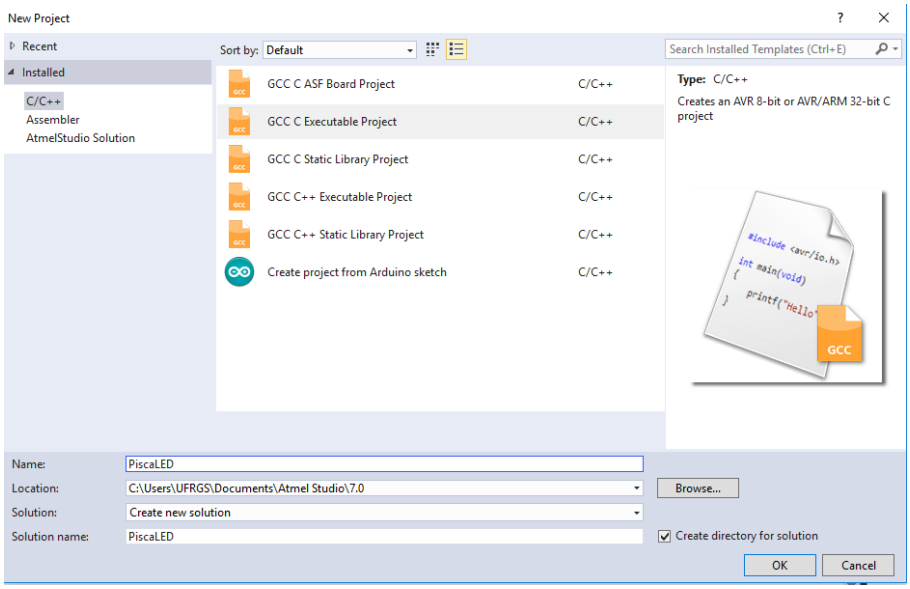
Programação

- Atmel Studio
  - Programação em C e Assembly
  - Depurador
  - Simulador (um pouco limitado)
  - Programador (diversos adaptadores e placas comerciais)

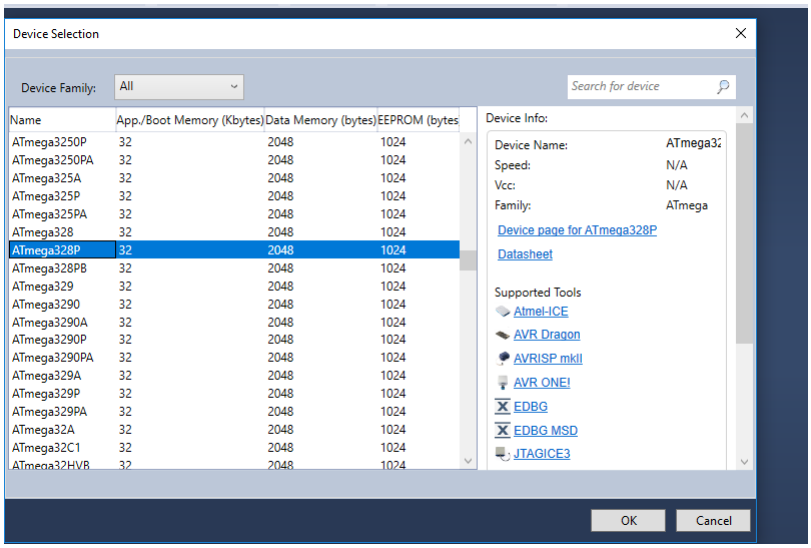
Programação



Programação



Programação



Programação

- Onda de 300Hz

```
Freq_LED 300 Hz
T_LED = 1/300 = 3,3ms => 1,66ms x 2

Prescaler de 256
Freq_timer =  16MHz/256 = 62500
T_timer =  1/62500 = 16us

Num_contagens = 1,66ms/16us ≈ 104
```

Tá dentro de um número de 8 bits, então ok

AVR Timer/Counter 0

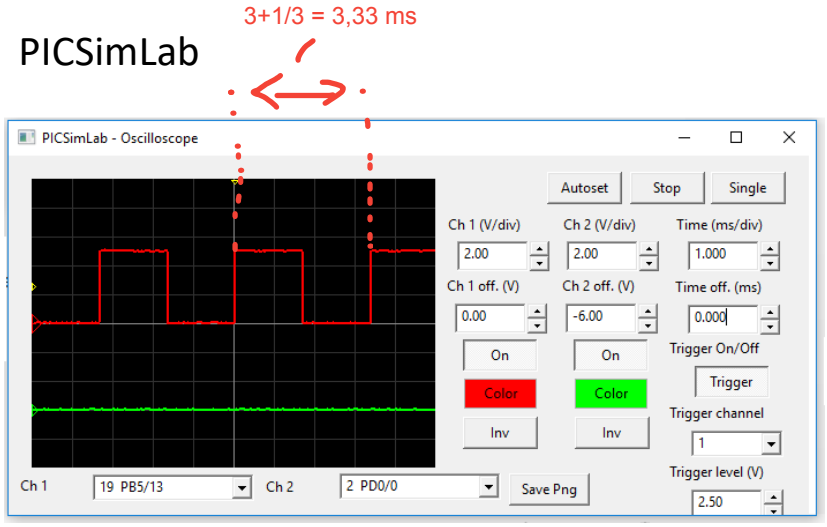
Programação (atraso 1,66ms p/ freq. 16MHz)

```
TCCR0A = 0;          // configura modo normal
TCCR0B = 0x4;        // configura clock /256
TCNT0 = 152;         // valor inicial
TIFR0 = (1<<0);      // limpa flag

while ((TIFR0 & (1 << 0)) == 0);   // aguarda flag
```

Quer 154 contagens, e ele estoura em 255+1, ou seja, é necessário 256-154 contagens

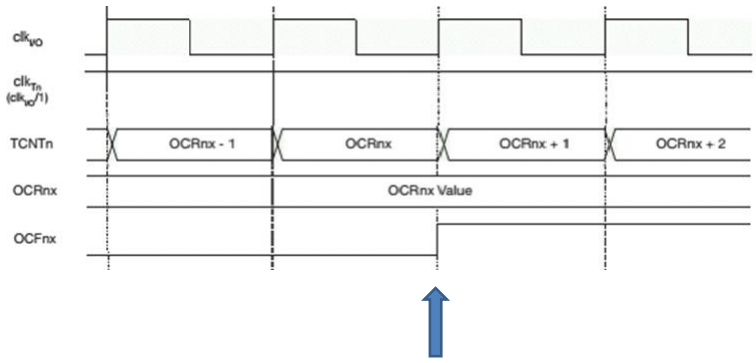
Simulador

- PICSimLab



AVR Timer/Counter 0

- Modo comparação (Clear Timer on Compare)

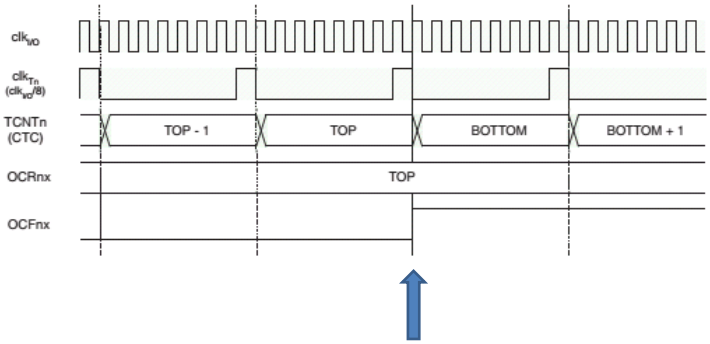


OCF0A é ativado quando
TCNT0 = OCR0A

AVR Timer/Counter 0

- Modo comparação (Clear Timer on Compare)

Exemplo com prescaler = 8



OCF0A é ativado quando
TCNT0 = OCR0A

Programação

```
#include <avr/io.h>

void atraso()
{
    TCNT0 = 152;        // Valor inicial (104 contagens)
    while ((TIFR0 & (1 <<0)) == 0); //  Aguarda flag de estouro
    TIFR0 |= (1<<0);    // Limpa flag de estouro
}

int main(void)
{
    DDRB =  (1 << 5);   // LED como saida

    TCCR0A = 0;         // Configura modo normal
    TCCR0B = 0x4;       // Configura clock/256
    TCNT0 = 152;        // Valor inicial (104 contagens)

    while (1)
    {
        PORTB |= (1 << 5);
        atraso();
        PORTB &= ~(1 << 5);
        atraso();
    }
}
```

(modo normal)

```
#include <avr/io.h>

void atraso()
{
    TIFR0 = (1 << 1);       // Limpa flag de comparacao A

    while ((TIFR0 & (1 << 1)) == 0);

}

int main(void)
{
    /* Replace with your application code */
    // LED no pino PB5
    DDRB = (1 << 5);    // Pino PB5 é saida (LED)

    TCCR0A = 0x2;       // Modo CTC para Timer 0
    TCCR0B = 0x4;       // Clock/256
    TCNT0 = 0;          // Zera timer
    OCR0A = 104;        // Valor da comparação 104 contagens (1,66ms)

    while (1)
    {
        PORTB |= (1 << 5);      // Ativa LED
        atraso();
        PORTB &= ~(1 << 5);     // Apaga LED
        atraso();
    }
}
```

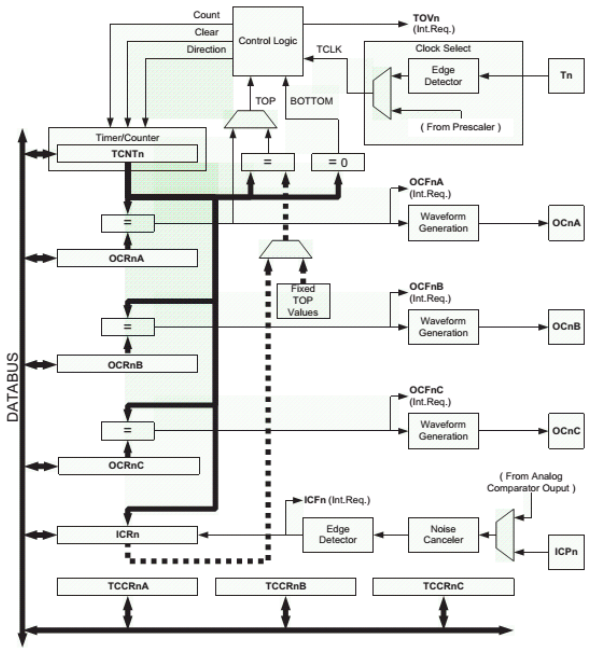
(modo ctc)

Não precisa recarregar o timer, pois é feito automaticamente e o bit de análise é o 1 (no while de atraso)

AVR Timer/Counter 1

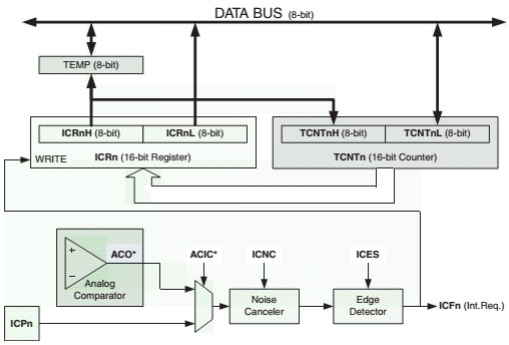
– Temporizador de 16 bits
  – conta de 0 a 65535 (0xFFFF)
  – Fonte de Clock interna ou externa
  – Captura de entrada por evento evento em pino ICP
- Prescaler
- Interrupção no Overflow
- Comparadores (A e B) para gerar forma de onda (PWM) em resolução 8, 9 ou 10 bits como contador Up ou Down

55



AVR Timer/Counter 1

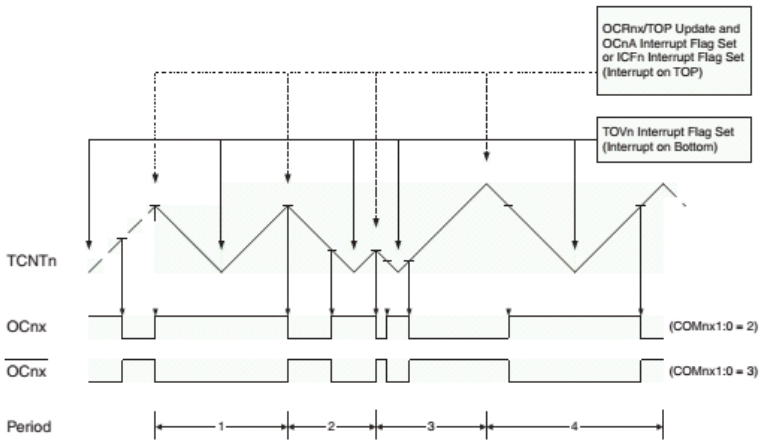
- Modo Captura



57

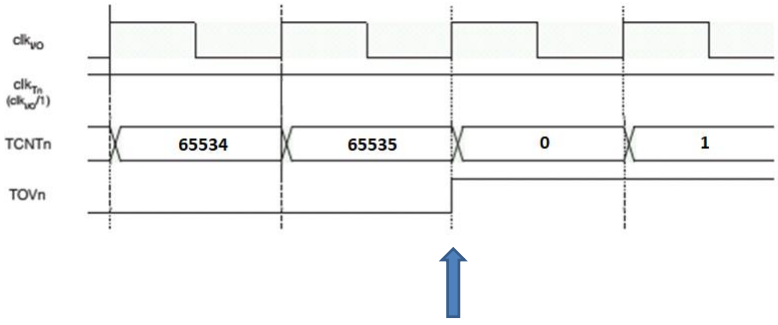
AVR Timer/Counter 1

- Modo PWM



AVR Timer/Counter 1

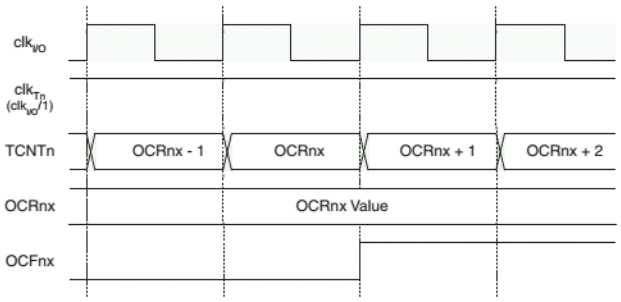
- Modo normal de operação



TOV1 é ativado após overflow

AVR Timer/Counter 1

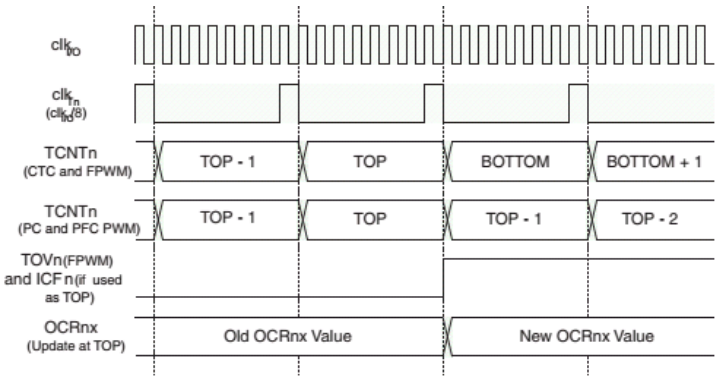
- Modo comparação



AVR Timer/Counter 1

- Modo comparação (Clear Timer on Compare)

  (exemplo com prescaler)



AVR Timer/Counter 1

TCCR1A

Timer/Counter Control A Register – TCCR1A

TCCR1A – Timer/Counter 1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

62

AVR Timer/Counter 1

TCCR1A

Timer/Counter Control A Register – TCCR1A

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

AVR Timer/Counter 1

TCCR1B

Timer/Counter Control B Register – TCCR1B

TCCR1B – Timer/Counter 1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

AVR Timer/Counter 1

TCCR1C

Timer/Counter Control C Register – TCCR1C

TCCR1C – Timer/Counter 1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOCnA: Force Output Compare for Channel A**
- **Bit 6 – FOCnB: Force Output Compare for Channel B**
- **Bit 5 – FOCnC: Force Output Compare for Channel C**

65

Programação

- Onda de 300Hz

```
Freq_LED 300 Hz
T_LED = 1/300 = 3,3ms => 1,66ms x 2

Sem Prescaler
Freq_timer =  16MHz
T_timer =  1/16MHz = 62,5ns

Num_contagens = 1,66ms/62,5ns ≈ 26667
```

Programação

```
void atraso()
{
    TCNT1 = 38869;      //Para 300Hz => 26667 contagens
                        //assim : 65536 - 26667 => 38869
    while ( (TIFR1 & (1<<0))==0);   // Aguarda flag de estouro
    TIFR1 = (1<<0);     //Limpa flag de estouro
}

int main(void)
{
    DDRB = 1 << 5;      //Pino PB5 eh saida

    TCCR1A = 0;         //Modo normal
    TCCR1B = 0x1;       //clock
    TCNT1 = 38869;      //Para 300Hz => 26667 contagens
                        //assim : 65536 - 26667 => 38869

    while (1)
    {
    PORTB |=  (1 << 5); //Ativa PB5
    atraso();
    PORTB &= ~(1 << 5); //Apaga PB5
    atraso();
    }
}
```

(modo normal)

```
void atraso()
{
    TIFR1 = (1 << 1);       // Limpa flag de comparacao A
    while ((TIFR1 & (1 << 1)) == 0);

}

int main(void)
{
    /* Replace with your application code */
    // LED no pino PB5
    DDRB = (1 << 5);    // Pino PB5 é saida (LED)

    TCCR1A = 0x0;       // Modo normal para Timer 0
    TCCR1B = (1 << 3) + 1;  // Clock direto + CTC
    TCNT1 = 0;          // Zera timer
    OCR1A = 26667;      // Valor da comparação (1,66ms)

    while (1)
    {
        PORTB |= (1 << 5);      // Ativa LED
        atraso();
        PORTB &= ~(1 << 5);     // Apaga LED
        atraso();
    }
}
```

(modo ctc)

Interrupções

Interrupções
- RESET
- Timers
- Hardware
  - Seriais
  - Pinos externos
  - Comparador analógico

Vetores de interrupção fixos

AVR Interrupt

ATMEL Studio

ISR(tipo_interrupção); // Função padrão para interrupções

sei(); // Habilita interrupções globalmente

Vector No	Program Address	Source	Interrupt Definition	Arduino/C++ ISR() Macro Vector Name
1	0x0000	RESET	Reset	
2	0x0002	INT0	External Interrupt Request 0 (pin D2)	(INT0_vect)
3	0x0004	INT1	External Interrupt Request 1 (pin D3)	(INT1_vect)
4	0x0006	PCINT0	Pin Change Interrupt Request 0 (pins D8 to D13)	(PCINT0_vect)
5	0x0008	PCINT1	Pin Change Interrupt Request 1 (pins A0 to A5)	(PCINT1_vect)
6	0x000A	PCINT2	Pin Change Interrupt Request 2 (pins D0 to D7)	(PCINT2_vect)
7	0x000C	WDT	Watchdog Time-out Interrupt	(WDT_vect)
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A	(TIMER2_COMPA_vect)
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B	(TIMER2_COMPB_vect)
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow	(TIMER2_OVF_vect)
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event	(TIMER1_CAPT_vect)
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A	(TIMER1_COMPA_vect)
13	0x0018	TIMER1 COMPB	Timer/Counter1 Compare Match B	(TIMER1_COMPB_vect)
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow	(TIMER1_OVF_vect)
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A	(TIMER0_COMPA_vect)
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B	(TIMER0_COMPB_vect)
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow	(TIMER0_OVF_vect)
18	0x0022	SPI, STC	SPI Serial Transfer Complete	(SPI_STC_vect)
19	0x0024	USART, RX	USART Rx Complete	(USART_RX_vect)
20	0x0026	USART, UDRE	USART, Data Register Empty	(USART_UDRE_vect)
21	0x0028	USART, TX	USART, Tx Complete	(USART_TX_vect)

Programação

```
#include<avr/io.h>
#include<avr/interrupt.h>

ISR(TIMER1_COMPA_vect)        // Interrupcao por comparacao A
{
    PORTB ^= (1 << 5);
}

int main()
{
    DDRB = (1 << 5);          // LED como saida

    TCNT1 = 0;
    OCR1A =  31250 ;          //Freq saida = 250Hz

    TCCR1A = 0x00;
    TCCR1B = 1 + (1 << 3) ;   //  CTC
    TIMSK1 = (1 << OCIE1A);   // Habilita interrupcao por atingir comparador A no timer1

    sei();                    // Habilita interrupcoes globais
    while(1);
}
```